A Review on Android Malware: Attacks, Countermeasures and Challenges Ahead

ShymalaGowri Selvaganapathy[1,*], Sudha Sadasivam[2] and Vinayakumar Ravi[3]

[1]*Department of Information Technology, PSG College of Technology, Coimbatore, India*
[2]*Department of Computer Science and Engineering, PSG College of Technology, Coimbatore, India*
[3]*Center for Artificial Intelligence, Prince Mohammad Bin Fahd University, Khobar, Saudi Arabia*
E-mail: shymalagowri@gmail.com
[*]*Corresponding Author*

Received 09 September 2020; Accepted 10 December 2020;
Publication 11 March 2021

Abstract

Smartphones usage have become ubiquitous in modern life serving as a double-edged sword with opportunities and challenges in it. Along with the benefits, smartphones also have high exposure to malware. Malware has progressively penetrated thereby causing more turbulence. Malware authors have become increasingly sophisticated and are able to evade detection by anti-malware engines. This has led to a constant arms race between malware authors and malware defenders. This survey converges on Android malware and covers a walkthrough of the various obfuscation attacks deployed during malware analysis phase along with the myriad of adversarial attacks operated at malware detection phase. The review also unscrambles the difficulties currently faced in deploying an on-device, lightweight malware detector. It sheds spotlight for researchers to perceive the current state of the art techniques available to fend off malware along with suggestions on possible future directions.

Journal of Cyber Security and Mobility, Vol. 10_1, 177–230.
doi: 10.13052/jcsm2245-1439.1017

Keywords: Malware, android, anomaly detection, attacks, defense, adversarial attack, evasion attack, obfuscation attack.

1 Introduction

The inverse relationship between malware and anti-malware has lead to the coevolution of malware authors and malware defenders for decades. The task of malware detection is a well-fought arms race between malware authors who continuously update themselves to deceive anti-malware engines and malware defenders who try to update and bring in better security mechanism to handle the various malware attacks [1–7]. Even as technology for countermeasures improves continuously, malware author designs malware with the objective of bypassing the anti-malware engines by formulating novel methods to evade detection and fulfill their malicious intent. Anti-malware engine can be utilized to comprehend the functioning of malware and its influence on the compromised node. The formulation of an anti-malware engine embodies two connected components: malware analysis and malware detection. Malware Analysis is the procedure of taking in the executable under inspection and performing reverse engineering techniques on it to extract features and determine the origin, functionality, and potential impact of malware [8]. Malware detection ingests those features and perceives whether the sample under inspection is a malicious or benign entity based on patterns or functionality [9].

The attacks against malware analysis framework are obfuscation attacks which tries to hinder the reverse engineering process by concealing the internal malicious functionalities and makes the analysis and detection harder [10–17].

Learning based solutions for malware detection [18, 19] are progressively adopted by anti-malware companies like Mcafee [20], Sophos [21], Cylance [22], CrowdStrike, [23] to name a few. Learning based techniques such as Machine Learning(ML) models and its subset Deep Learning(DL) models are under the constant threat of well-crafted cyber-attacks known as adversarial attacks, thus making it a fragile component in the security pipeline [24]. Thus, the need to develop secure learning techniques which are immune to adversarial attacks is on the rise and is emphasized by many government intelligence agencies and technology sectors. DARPA's Guaranteeing AI Robustness against Deception (GARD) [25], Google's Responsible AI Practices [26], IBM's Adversarial Robustness Toolbox (ART) [27], Microsoft's Securing the Future of Artificial Intelligence and ML at Microsoft [28], SecML [29] and

Cleverhans [30] are few attempts to counter the adversarial attacks on ML models on the image space. The available comprehensive literature review articles on adversarial attacks against ML models and possible defenses in the image space are [31–36] to name a few.

Sophos Threat Report for the year 2020 particularizes on the rise of attacks against ML models deployed for malware detection [37]. Cylance's PROTECT anti-malware engine which deploys DL models has been recently evaded by adversarial attacks [38]. Thus, ML based malware detectors are under the constant threat of adversarial attacks [39–45]. This demands for the development of robust and secure learning models to be deployed in malware detectors which can withstand the adaptive adversarial attacks [46–52].

The progress made in the analysis and detection of malware have been discussed in detail in recent review articles like [14, 19, 53–57]. Adversarial attacks and possible defense for malware space is reviewed in [47, 58]. The available literature survey articles have not addressed the malware space problems in an all-inclusive manner. This has given rise to the need for analyzing the current available approaches and to identify their shortcomings and propose possible path for future contributions. This review article tries to provide an end to end spotlight on the difficulties encountered in building an effective and efficient malware detector for the Android domain. There are now more than two and half billion active Android devices as reported by Google in 2019 [59]. In its threat report for the year 2020, Mcafee affirms the tremendous increase in mobile malware attacks in the recent years along with an observation on how attackers are improvising their strategies for defeating detection [60]. Android malware are on the rise due to Android's open source nature. The feasibility to download and install Android applications from unverified sources makes it easy for malware authors to bundle and distribute applications with malware. The rapidity in which malware evolves in terms of volume and diversity makes it harder to be detected effectively.

The contribution of this review article are as follows:

1. Summarizes the possible attacks and current defenses that exist in the Android malware threatscape.
2. Poses few research questions to encounter the challenges faced while building an effective and efficient malware detector while responding to the formulated research questions elaborately.

The rest of the review article is organized as follows: Few research questions are devised in Section 2. Background on malware and malware variants is given in Section 3. Section 4 discusses about the generic anti-malware

engine. Section 5 provides a brief overview of the factors to be considered when designing a malware detector with emphasis to static analysis and lightweight feature extraction. Various artifices of the malware author along with possible attacks encountered by an anti-malware engine is covered in Section 6. Section 7 elucidates the techniques available for increasing the defense of the learning based systems and Section 8 concludes the survey with possible future research directions.

2 Research Questions Framed: Android Malware Dectection

The subsequent sections unravel each of the following research questions in detail. These research questions enable to provide a wider and deeper understanding of the overall Android malware detection.
RQ1: What are the Android malware variants? How can they be detected effectively?
RQ2: How does an anti-malware engine work?
RQ3: How to collect Android apks from the wild?
RQ4: What tools are available for reverse engineering an Android apk?
RQ5: Which learning based technique is suitable for malware detection?
RQ6: What kind of attacks are to be dealt with when performing malware analysis and detection?

3 Background on Malware and Malware Variants

As stated by Denning, it is strenuous to protect computer system and networks from intrusions [61]. Bace [62] describes intrusion as an attempt to breach the Confidentiality, Integrity and Availability so as to bypass the security mechanisms of a computer or network. Intrusion Detection and Prevention Systems (IDPS) primarily focus on detecting possible intrusions, attempting to stop them and reporting them to security administrators [63]. Based on the location of the information sources, IDPS can be categorized as host based or network based approach [64]. Host based Intrusion Detection System (HIDS) checks for suspicious activity at the host level [65]. A Network based Intrusion Detection System (NIDS) checks for intrusions in the network traffic at gateway and routers by analyzing network protocol activities [66]. Endpoint security solutions for smartphones, workstations, servers

and other computing resources have to offer protection from malicious attacks. Malware are unwanted software which imparts multi-sided concerns such as sneaking, stealing, causing financial loss or gaining control without the consent of the end-user. The following subsection introduces its readers to a brief history of malware and its variants.

3.1 History of Malware

The emergence of computer virus was conceived around 1948 when John von Neumann extended Alan Turing's automata [67] by incorporating the theory of self-reproducing automata [68]. He outlined the way in which a computer program could replicate itself. Bob Thomas from BBN technologies wrote a maiden self-replicating program, the Creeper system [69] to assess the theory given by von Neumann. Fred Cohen affirmed von Neumann's postulates about computer viruses and explored malware characteristics like detection and obfuscation [70]. Cohen phrased the term computer virus as a program that can contaminate other programs by forming mutants of the original program [71]. Since then, computer virus was the keyword used to group the various malicious software including those that did not mandatorily infect other software.

In 1990, Yisrael Radai pioneered the term malware to denote virus, worm, trojan and other similar malicious entities [72]. Subsequently, the term malware is used as the representative to denote any software which exhibits any malicious activity. The journey of mobile malware [73] started around the year 2000 when smartphone's based on the Symbian Operating System (OS) was introduced; Cabir worms [73] and Timofonica [74] were some of the few malware which initially emerged and started causing havoc. Google's Android OS came in 2008 and it started attracting application developers as well as malicious authors due to its open source nature. The SMS.AndroidOS.FakePlayer.a trojan [75] was the first Android malware to be detected in 2010. Since then the volume and veracity of Android malware has had a tremendous increase.

3.2 Understanding Malware Variants

Malware are characterized according to their propagation procedures and their activities accomplished on the infected system [76]. Figure 1 illustrates the categorisation of malware along with the possible diverse cyber security breaches. The categorization of the malware types can be centered on the

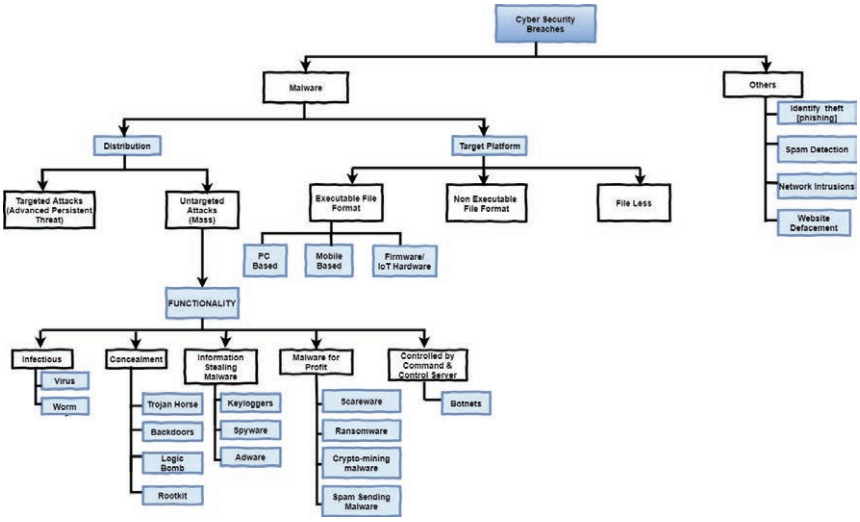

Figure 1 Categorisation of malware along with the possible diverse cyber security breaches.

attacker's objective into conventional or mass malware and targeted or smart malware [77]. Mass malware attempts to affect any victim machine and cause damage as much as possible. These are less sophisticated and can be defended easily. Targeted attacks or smart malware are Advanced Persistent Threats [APT] [77] which utilize cutting edge cyber-security techniques and puts in consistent and continuous efforts to gain access. It tries to remain undetected for long term and causes substantial damage to the specific target.The mass malware threat landscape constituting diverse malware avatars with various capabilities can be roughly categorized based on the functionality to the following groupings [78].

3.2.1 Infectious

Infectious malware are those that spread to infect new victims. Malware spreads through disks, e-mail, messaging services, Internet protocols, compromised legitimate websites, advertisements, to name a few. Infectious malware can be categorized as:

- Virus:
  Analogous to the biological virus which infects a living cell and takes control, the computer virus is a code fragment which necessitates a host program to execute its functionality [76]. Virus inserts its malicious contents into a legitimate program and infects it. When the host program

is executed, the parasitical program takes control, executes its payload and reveals its contemplated behavior. It ubiquitously propagates by infecting applications executing on the host node and other nodes accessible through network connections. A few examples for cross platform mobile virus are Crossover, Dust, Lasco and CardTrap [78].

- Worm:
  A worm [76] is an unallied and self-reliant program which propagates and replicates by spawning itself on infested nodes and explores further vulnerable nodes via accessible network and copies itself on these nodes. Some of the worms available for mobile are Cabir, ZeuS MitMo, Beselo and CommWarrior.

3.2.2 Concealment

Concealment malware are those that takes measures to hide its detection. They employ persistence mechanism to make them undiscoverable. They can be categorized as follows:

- Trojan Horse:
  In general, a trojan [76] is any malware that deludes its authentic purpose. To bait the innocent user, they disguise and advertise as legitimate entities with alternate useful functionality but their payloads perform malicious activities. User assistance for installing and executing the infested program is imperative for the infection to take place as Trojans neither spread nor require any host program to work. A few trojans which created havoc in smartdevices include DownAPK, GatSPY, MasterKey, Fontal, Liberty Crack and Infojack [54].
- Trapdoors or Backdoors:
  Backdoors [76] are malware which surreptitiously gain access to entities such as system or processes in a fabricated way that is not supported by its designers. The typical payload of a worm or a trojan is to setup a backdoor so attackers would gain control in an unauthorized manner to corrupt, delete or steal sensitive contents. A few examples include Brador which is a backdoor targeted for pocket PC hand-held devices with Windows mobile OS [79], Twitoor [80] which is an Android backdoor and botnet that imitates like a MMS app and accepts commands via tweets.
- Logic bombs:
  Analogous to a delayed-action bomb whose detonation could be delayed, logic bomb [76] or slag code are code fragments programmed with malignant purpose which remain inactive in the host. The code gets

triggered when specified criteria is satisfied and gets executed to cause the expected sabotage to the host. This attack is usually inflicted by an insider having privileged access. Remote Control System Android (RCSAndroid), a spying malware stealthily waits for SMS messages from certain contacts [81]. Holy Colbert waits for reboot of the device and then checks for device date against the specified hard coded date. If the dates match, it starts sending spam messages to contacts [82].

- Rootkits:
  Rootkits [83] are a set of software tools typically used by a malicious actor to gain and maintain privileged access to a compromised system in an unauthorized manner. A rooted host is one with a rootkit installed on it. The rootkit facilitates remote access to the malicious actor to dispatch further attacks or collect intelligence. The malicious actor executes clandestinely without the user's permission as rootkits are equipped with sophisticated techniques to alter system files to be undetected. The rootkit, DroidDream [84] injected few root exploit activities in some of the Google Playstore apps in 2011. HummingBad rootkit installs fraudulent apps and performs clickfraud to generate revenue for its parent company, Yingmob [85].

3.2.3 Malware for stealing information

These malware infringe to steal sensitive information and misuse them against the user. They are categorized as

- Keyloggers:
  Keyloggers [11] records the keystrokes of a user which may include sensitive information such as banking credentials, login details and exposes them to the attacker. Mysterybot which is a keylogger, ransomware and a banking trojan records the location of the user's touch gesture screen position and predicts the corresponding key pressed on a virtual keyboard [86]. Mspy,a spyware and a keylogger records all keystoke usage on the smartphone [87].
- Spyware:
  Spying software [88, 89] usually violate user's electronic privacy by illegally inspecting user activities. Surreptitiously, such software try to steal sensitive personally identifiable information, passwords, financial details which can be used for illicit financial benefit. They stealthily collect clickstream data, keystrokes, browsing behavior and transmit them to the attacker when connected to the Internet. Spyware do not self-replicate and are usually payloads of Trojans. Flexispy monitors

the target device activities by intercepting emails, SMS, inspects Social media apps and listens to calls [87].

- Adware:
  Similar to spyware, adware [88] illicitly collect and transmit user preferences but are primarily used for marketing purposes. Activities such as displaying hand-picked commercial advertisements, redirecting user's browser to intended sales websites are performed to generate revenue for attackers. Few unethical adware apps are Car Racing 2019 and Mobnet.io Screen Stream Mirroring [90].

3.2.4 Profit seeking malware

Malware can also be used to create revenues for the attacker. They are categorized as follows:

- Scareware:
  Scareware deceive users into purchasing needless software by inducing some form of fear through fake alerts. AndroidDefender is a scareware and fake mobile anti-virus providing fake alerts and insists user to buy their product to protect the end user device [91].
- Ransomware:
  Ransomware [92] are coercion software that either encrypts file contents or locks user access out of the system and ordains the user to pay a ransom to release hostage of the computing resources. Some ransomware families like Maze, Sodinokibimb forces the victim to pay the ransom by threatening to disclose the data [93, 94]. Android ransomware like Filecoder [80] are disseminated through online forum links, spreads to device contacts through SMS with malicious links, encrypts device contents and demands ransom. other notable Android ransomware include WannaLocker, LeakerLocker and Xbot [95].
- Spam-sending malware:
  These malware utilize the infected machine to send spam messages and tries to generate revenue for the attackers [11].
- Crypto-mining malware:
  Malice crypto-mining or Cryptojacking is an emerging profit seeking malware attack which does not damage infected device or steal infected device data. It instead loots the resources of the infected device like CPU processing power to convert computing resources into digital electronic money called cryptocurrencies by performing mining operations which are complex mathematical encryption operations. Attackers misuse the infected devices without the victim's consent by stealing their

computational resources by executing complex scripts to mine cryptocurrency. Unlike ransomware, which threaten user's and gain profit, cryptojacking software conceals itself from the victim's attention. The victim suffers the damage caused by cryptojacking malware like device performance slow down, consumption of electricity and device life-time reduction [96]

3.2.5 Botnet

Resembling the backdoor, where an attacker can access and operate on the specific infected machine, all systems infested with the same botnet take in the same instructions from a sole command and control server [11]. The compromised machines or zombies are utilized to launch attacks against other computing resources such as bringing down a legitimate website using distributed denial of service attacks, spreading spam contents and performing click frauds. Geost botnet can access user personal information, control the infected device by redirecting the network traffic,send SMS and perform bank communications [97]. In Chaois [98] the infected device user is tricked into SMS scams and unknowingly pay cybercriminals through premium SMS fraud tricks. Twitoor [99] is an Android backdoor and botnet which imitates like a MMS app and accepts commands via tweets.

The evolution of malware into its innumerous avatars have now recently reached the fileless malware approach also known as Advanced Volatile Threats (AVTs) [100, 101]. AVTs execute its malicious activities in the volatile main memory and is able to escape anti-analysis techniques as the infection traces are available only in memory during its execution. It tricks common tools like PowerShell, Windows Management Instrumentation, command prompt, .NET framework, Remote Desktop Protocol (RDP) into attack vectors [56, 100–102].

Fileless malware utilize scripts like JavaScript, Visual Basic for Applications script and compiled HTML files under the hood of a system process [103]. Currently, fileless malware work with Windows OS platform, but security analyst are predicting that it will slowly creep into other platforms in the near future [104].

4 Anti-Malware Engine

An anti-malware engine intakes an executable and segregates it as malware or not by utilizing two stages namely malware analysis and malware detection as given in Figure 2.

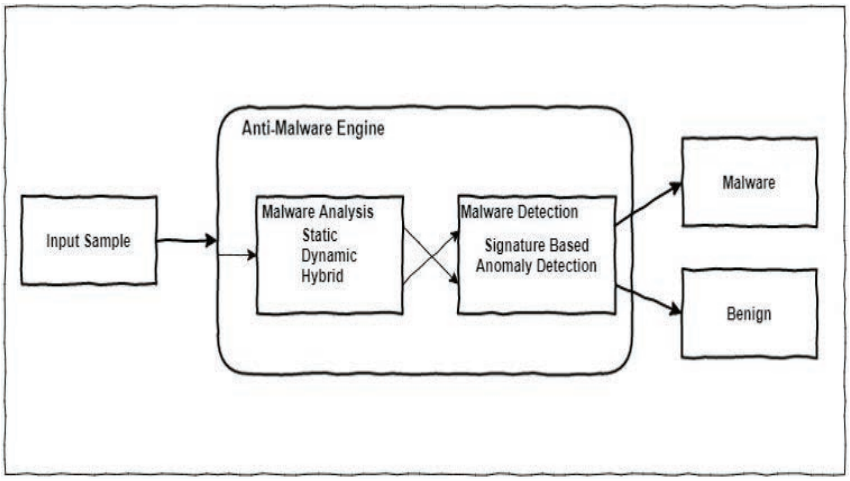

Figure 2 Anti-malware engine.

Malware Analysis is employed to understand the risk and intentions of malware [8]. Malware analysis can be categorized into static, dynamic and hybrid approaches. Static analysis also known as structural analysis or code analysis [8] dissects and extracts features without executing the code. These techniques have high code coverage and can provide faster interpretation of the sample and are easily scalable. But, they face the mammoth drawback of the inability to resolve malware utilizing obfuscation techniques [105]. Obfuscation mechanism are those that obscure the reverse engineering process by applying procedures such as encryption, compression, dynamic linking, dead code insertion, polymorphism, to name a few. Dynamic analysis or behavioral analysis technique [19] can overcome these issues by executing the sample in a constrained environment such as a sandbox or an emulator for a limited period. The behavioral parameters such as API calls, instruction traces, registry changes and memory writes are observed. This resource intensive technique has the drawback of producing a single path code coverage and could be vulnerable to malware utilizing system-call obfuscation techniques and anti-analysis techniques [19]. Hybrid methods exploit the advantages of both static and dynamic approaches by utilizing both the techniques together. Features from static analysis and dynamic analysis are combined to perform hybrid analysis [1, 106].

Malware detection could be zoned into signature based methods and anomaly based techniques. Signature based or misuse detection methods

encompasses a repository of known malware signatures against which it compares the incoming samples to categorize them. The huge hindrance in this traditional technique is that it needs to keep the repository up to date [8, 9, 107] else it cannot detect unknown samples. Anomaly based also known as heuristic based detection techniques make use of the behavioral approach of learning. It comprehends the normal patterns and any deviations observed are designated as malicious. However, not all abnormal activities are malicious, so it is prone to false positives. This approach has the benefit of detecting unknown malware and zero day attacks. Zero day attack targets security flaws that is unknown or unpatched by the software vendor [2]. Anomaly detection based security solutions [19] can deploy expert systems, string matching, state modeling,rule-based systems, genetic algorithm, immune system models and learning based techniques. Learning based models have superior performance when trained appropriately using large volume of data. They can extract better patterns by utilizing deeper networks thereby reducing false positives [19].

5 End2End Framework for Android Malware Detection

The end2end framework for Android malware detection is given in figure 3 which involves three phases such as dataset collection, malware analysis and malware detection.

5.1 Data Collection

For effective malware detection, Android applications comprising a balance of benign samples and malicious samples should be collected. Data collection should be a representative of the various malware families in the wild. Table 1. gives a picture of possible sources where apks can be collected and assimilated to form a comprehensive dataset. Android applications executed on the Android operating system are packaged as Android Application Package(.apk) archive files. The APK files are digitally signed with the application developer's certificate. Figure 4 adapted from [108] illustrates the major directories and files encompassed in the .apk file which are explained as follows:

- AndroidManifest.xml holds the permissions, application version and libraries used by the application. This core manifest file is in the Android binary xml format.

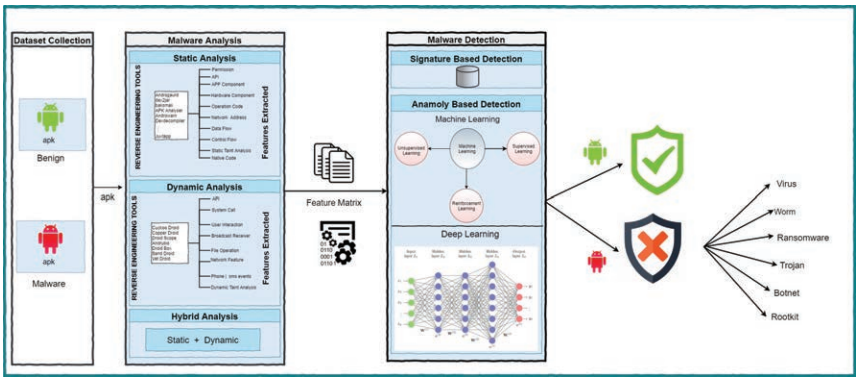

Figure 3 End2End framework for android malware detection.

- LIB folder contains native libraries which are shared non-java libraries loaded by the Java Native Interface(JNI) at run time. Processor specific and platform dependent compiled code are placed in the LIB folder.
- META-INF folder contains the signatures files CERT.RSA, CERT.SF and the MANIFEST.MF.
- resources.arsc contains the precompiled compressed resources.
- RES folder contains resources which have an ID assigned during compilation.
- ASSETS folder contains application assets like media files, fonts, HTML contents which are accessible through the Asset-Manager class.

5.2 Malware Analysis

This section brings to light the steps involved in understanding how an executable can be converted to a format that can be given as input to a learning based algorithm. The executable is converted from machine language format to interpretable format using reverse engineering tools, from which features are extracted to construct either tabular or sequential based features for the learning based techniques. The rest of this subsection explores the commonly available feature extraction methods, followed by exploring techniques which emphasize on lightweight methods for feature extraction. Options available for training and deploying the model are also articulated in this section.

5.2.1 Reverse engineering

Reverse engineering techniques take in the binary instructions and reveal the working logic of the inspected program by utilizing various tools and

Table 1 Android dataset sources

S. No.	Dataset	Description
1	Rmvdroid [109]	9,133 malware samples
2	Android PRAGuard [17]	10479 samples
3	AndroZOO [110]	Millions of samples
4	F-Droid [111]	Repository for Benign samples
5	appsapk [112]	Repository for Benign samples
6	apkpure [113]	Repository for Benign samples
7	DroidFusion [3]	5,560 malware+ 9,476 benign
8	CICInves And Mal2019 [114]	426 malware + 5,065 benign
9	AMD Project [115]	24,553 malware samples
10	CIC And Mal 2017 [116]	4,354 malware + 6,500 benign
11	CICAAGM dataset [117]	250 Adware + 150 General Malware + 1500 Benign
12	Android Botnet [118]	1929 android botnet
13	Drebin [119]	5560 malware + 123453 benign
14	ContagioDump [120]	189 malware samples

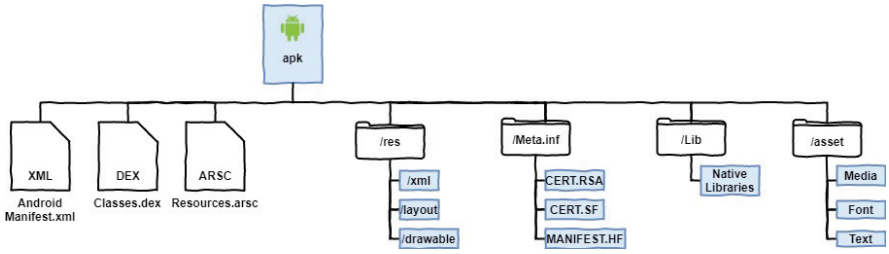

Figure 4 APK structure.

frameworks. A myriad of features can be extracted by feeding the samples from the collected dataset into the tools for reverse engineering the apks.

The extracted features from static malware analysis can be represented as a sparse binary feature matrix where columns represent the various features extracted. The presence or absence of a particular feature can be identified by checking the binary value at the particular index position of the feature vector as either a '1' or '0'. Dynamic analysis results in providing a sequence of API calls invoked with respect to various system related operations such as memory, process, file and network activity.

In static analysis, the Android APKs are unpacked, decompiled and inspected for malicious contents without executing them. Static features are extracted from the Android-Manifest.xml and the DEX code file. Although fast, it can be easily fooled by malware making use of obfuscation techniques. Androguard [121], apktool [122], dex2jar [123], radare2 [124], dexter [125] and IDAPro [126] are few static analysis tools available for converting the .apk format to an interpretable format for analyzing and extracting features. RemNux [127] and Android Reverse Engineering (A.R.E) [128] are few reverse engineering frameworks readily available for malware analysis.

The design process of an anti-malware engine should take into consideration the presence of obfuscated malware. The designer of the anti-malware engine should explore various obfuscation techniques that could be deployed and use appropriate methods to overcome obfuscation based attacks. ProGuard [129], DexGuard [130], DexProtector, [131] are few tools available for working with Android obfuscation techniques.

5.2.2 Feature extraction

The various features that can be extracted by malware analysis for Android applications are:

- Permissions used [106, 132, 133]
  An Android application indicates the requested permissions it needs before installation. The applications gets installed only when the user grants these permissions. Android security framework uses this method to enforce a permission based model to restrain admittance to end-user personal contents.
- Application components [134, 135]
  An Android application has four application components encompassing Service, Activity, Content Provider and Broadcast Receiver. Service components execute in the background without user interaction. Activity components furnish user interfaces. Exchange of data across applications is facilitated by using Content Providers. System-wide announcements are handled by Broadcast Receivers.
- Filtered intents [136–138]
  Intents are messages sent between components used by Android message passing system. Each component's action is indicated through the filtered intents. Intent-filters are used by each of the application component to register itself to obtain the Intents.
- API calls [139, 140]

API calls are used by the application to work with the device in order to accomplish the core functionalities.

- Network addresses [141–143]
  Network addresses can be used to access the URLs, IP addresses used by the application.
- Opcodes [142, 144]
  Operation Codes, their frequencies and opcode sequences used in the apps can be accessed.
- Native code [16, 17]
  Native code are function calls invoked on external binaries including shared libraries.
- Hardware components [119, 145, 146]
  Application access to hardware components such as camera, USB, GPS can be extracted.

Apart from the above specified features which are directly extracted from the .apk file, few techniques have been developed to extract graph based data structures for better and efficient malware analysis which encompasses:

- API Call Graph (CG) [147–149]
  Api Calls made between modules are marked as a directed graph with the nodes representing each API. Edges denote the directed links between the caller and the callee.
- Control Flow Graphs (CFG) [133, 150]
  The set of all paths traversed during program execution can be represented using a Control Flow Graph. The nodes can represent basic building blocks and edges correspond to transitions from one basic block to another using jump statements or branching statements. A basic block comprises of a set of sequential instructions with no branch statements.
- Data Flow Graphs (DFG) [149]
  Data Flow Graph can be used to visualize and observe the flow of sensitive information across the application between different entities.

The above techniques are considered as heavy weight approaches for feature extraction. They perform offline detection of malware with no constraints in regard to computing capabilities or handling huge scale of samples for data processing when compared to the energy and computation resources restrictions available on mobile device. This makes them harder to be considered as on-device detection models instantaneously. This has given rise to the need for identifying features which can be

retrieved in a light-weight manner while providing efficient detection of malware.

5.2.3 Light-weight feature extraction

Few models have been developed keeping in mind the requirement of lightweight analysis [5, 12, 13, 151–153]. These techniques try to understand the raw extracted multitudinous features and select only those features that help in discriminating malware better and keeping the analysis simple.

RevealDroid [13] identifies malware along with its possible family category. Lightweight feature subsets utilized in Revealdroid includes Android API usage at package-levels (PAPI), Android API usage at method-levels (MAPI), resolution of APIs caused by reflection calls and function calls carried out by native binaries. Usage of these features enable Revealdroid to achieve obfuscation resilience along with enhanced performance when compared to its counterparts deploying complex program analysis or methods utilizing huge feature sets. Revealdroid tends to expand its capabilities on exploring rooting malware and cryptocurrency mining malware.

LightDroid [12] bestows a lightweight malware detection which can be effectively implemented on mobile devices but with the trade-off of providing accuracy at an acceptable margin. It extracts a group of picture-based features by considering the raw pixel representation of first thousand bytes from the Dalvik executable files along with a nominal set of features from the AndroidManifest file. It fits a Convolutional Neural Network (CNN) model and has a comparatively faster detection at inference time when compared with RevealDroid [13] and Drebin [119]. Lightdroid finds obfuscated malware by incorporating a computationally complex method into its training phase and yields a lightweight inference phase model deployable in mobile devices. The article constructs another obfuscation resilient model named ORDroid which detects transformed and obfuscated malware samples. It uses a different set of features comprising the PAPI, MAPI and reflection features from RevealDroid along with sequences of opcodes from the DEX file. ORDroid builds a Gated Recurrent Units (GRUs) model which has performance better than Revealdroid and Drebin.

A methodical investigation on feature engineering is performed in DroidSeive [152] to build a heterogeneous feature set capable of adapting to different malware yet being lightweight and computationally reasonable. It constructs a map relating the classes of extracted features comprising syntactic and resource centric features. Resource centric features take into

account the artifacts and footprints left behind by hiding or repacking a malicious content. Syntactic features build on API calls, permissions and derive features from meta-information and explicit intents. ExtraTrees are used to build up the model. DroidSieve performs malware detection and family classification. Dataset utilized involves combination of obfuscated and non-obfuscated malware samples. DroidSieve is capable of detecting obfuscated malware utilizing native code, encryption and reflection. DroidSieve reflects how methodical feature engineering can yield better learning based malware detectors.

Tinydroid [151] employs an instruction simplification technique to build a lightweight static ML detection model which extracts opcodes from all the Dex files in an Android application. Tinydroid includes reflection API features and function calls to identify native binaries invocation as part of its features. It constructs simplified symbolic instructions by taking assistance of N-gram technique and feature reduction by using information gain technique. Affinity propagation clustering method is used for exemplar selection to reduce the number of training samples. It uses Random Forest, Support Vector Machine and Naive Bayes for classification and compares its performance with current AV tools. Tinydroid utilizes malware samples from Drebin dataset and benign samples from Google Play market.

1D-CNN [5] does not utilize any feature engineering or specific set of features from the apk file. Instead it considers the last 512 bytes to one kilo bytes of the raw APK which is converted into an image and fed into an one-dimensional CNN. The dataset comprises malicious samples from AMD [121] and Drebin [119]. Goodware samples are collected from Appsapk [112]and Apkpure [113] The performance of the model is not clearly explained by the author.

DroidKin [153] addresses the problem of identifying similarity between cloned and plagiarized applications. DroidKin develops a lightweight and robust approach utilizing meta-data and N-gram based opcode sequences from dex files to infer the relationships among similar applications. Simplified Profile Intersection (SPI) metric has been proposed as a similarity measure for evaluating source code author profiles through frequency analysis. Relationship between related applications such as twins, siblings, step-siblings, false step-siblings and cousins are identified based on the decreasing order of closeness. Droidkin applied few obfuscation transformations and analyzed its corresponding impact on the similarity determination of how and why applications could be related to each other.

5.2.4 Feature extraction for on-device lightweight testing applications

On-Device lightweight static learning based malware detection techniques usually train a model off-line in a high end system. As training with huge datasets utilizing DL models is resource intensive and time consuming; this is usually difficult with mobile devices which have limited capability. The optimal trained model with its fixed parameters after fine tuning is then transplanted into the on-device application for prediction purposes.

IntelliAV [154] is a freely available and deployable on-device lightweight learning based anti-malware for Android devices built using TensorFlow library. Lightweight features comprising of APIs from the Dex file, intent filters, permissions and application component statistics are extracted from the Manifest file to build the feature vector which comprises 4000 features. IntelliAV used 9,664 malware and 10,058 benign samples collected from VirusTotal and builds a model using TensorForest which is an ensemble based Random Forest model in TensorFlow. The trained model after fine tuning its performance is embedded on-device in the IntelliAV Android application. IntelliAV application offers QuickScan and CustomScan capability to scan all installed apps and downloaded/dropped APKs respectively in real time and associate a risk score for each. For testing an application on the device, IntelliAV extracts the application's features and feeds it to the optimized model which computes the risk associated with the application. IntelliAV claims to have low overhead on real devices and can analyze an application in less than 10 seconds. IntelliAV is a good start as an on-device anti-malware approach which can in future become more robust against adversarial ML attacks and pervasive malware obfuscation attacks.

Drebin [119] is a famous lightweight detector which does a linear sweep of an Android application using the Android AssetPackaging Tool and extracts around 5,45,000 different features from the manifest and dex files. The features are embedded in a vector space indicating presence or absence of a feature in an application. A linear SVM model is constructed by offline learning using 5560 malicious and 1,23,453 benign applications and the optimal learned model is transferred to the mobile device. The Drebin working model is not available publicly. The results mention that Drebin takes around ten seconds to scan an application and calculate its risk score. Drebin tries to address obfuscation attacks and code loading by monitoring the relevant loadClass and getInstance APIs. It discusses about Drebin's possible future extensions to address the inherent possibilities of an attacker performing mimicry attack and poisoning attack on its learner, handling reflection and

encryption attacks, utilization of better representative samples for its training dataset with frequent retraining using sanitized datasets.

Qualcomm's Snapdragon Smart Protect [155] is an on-device holistic ML based anti-malware commercial product whose specific design or feature set used are not publicly available.The model decides dynamically the optimal set of features to consider for the learning and the model is said to have a unique characteristic of the ability to prune the classifier to a lean classifier which works on this reduced feature set. The model is said to have good detection of zero-day malware and reflection based malware.

5.2.5 Feature extraction for on-device lightweight training and deployment apps

Previous section discusses on-device malware detectors that work by training the learning model in a dedicated server which is then embedded to the mobile device. This section discusses on on-device training which is usually challenging owing to the resource limitations inherent in the mobile devices.

Yuan introduced BL-AMD [4], a broad learning based lightweight on-device Android malware detector which can be either completely or incrementally trained directly on-device. Broad Learning (BL) enables fast training speed as it utilizes one shot computations for its parameter calculation. It provides better generalizations by making use of sparse autoencoders and hierarchical feature transformations. Incremental learning facilitates model retraining with new samples. A total of 379 static features are extracted from API calls, intent actions and permissions to construct a binary feature vector of an app which is then fed to a three layer neural network. The first two layers are data-processing units comprising feature layer and enhanced layer which extracts representations of the data provided to them. The third layer which is the output layer processes the outputs of both feature layer and enhancement layer to provide the probabilities of the app to be malicious and benign. The results mention that BL-AMD scans applications averagely in about 0.2 seconds. The model tests its robustness to black box adversarial attack MalGAN [156] and demonstrates better performance achieved through its incremental retraining capability.

5.3 Malware Detection

Commonly used ML and DL models have been successfully used for Android malware detection [19,55]. Few architectures improve the classification accuracy by making use of the distinct properties of the domains in which they

are used. For computer-vision applications, Convolutional Neural Networks architectures are commonly used as they are better on input containing translation invariant properties that are found in images. Recurrent Neural Networks are commonly used in hand-writing recognition and natural language processing as they perform better with input that has to be processed sequentially. The binary indicator vector used to represent the Android applications does not possess the structural properties that are present in the above mentioned applications. So, Deep Neural Networks that are based on regular feed forward neural network architecture can be used for malware classification. The model has to be evaluated in a temporally consistent manner for better generalization capabilities.

6 Artifices of Malware Author

While manual analysis of malware and successive construction of signatures is at stake, malware authors tend to generate a number of different malicious samples from a single malware. This section discusses the attacks that can be launched against an anti malware engine. It considers obfuscation attacks which are employed during malware analysis and adversarial attacks on the ML models deployed in the malware detection phase.

6.1 Obfuscation Attacks – Disturbance caused during Malware Analysis

Given the tremendous upsurge of malware, the necessity to develop automated analysis and detection techniques with least human intervention is the need of the hour. Every day, there are new malware detected even though in reality only few thousands of malware families may exist. Increase in creation of malware variants by the authors is motivated by the need to evade malware detection. Malware authors continue to expand their threat vector by scaling up on the types and functionality of the malware along with the increase in the volume of the new malware samples being generated. The end goals such as evading detection and generating new variants from existing samples are achieved by the usage of obfuscation techniques. Obfuscation techniques are used to deliberately convert readable code into obscured code while retaining the code functionality. These techniques were initially developed to protect from pirates who pried protected software and stole intellectual properties of software developers [157].

Adversely, malware authors started utilizing obfuscation techniques to conceal the internal malicious functionalities and to make analysis and

detection harder [10, 11]. Obfuscation techniques utilized by Android malware exploit the open source nature of Android framework [16, 17]. These anti-analysis techniques are usually categorized based on the malware analysis technique to be invalidated. To deter static analysis, obfuscation techniques like compression, encryption, code obfuscation, anti-disassembly and polymorphic techniques are engaged [14]. To thwart dynamic analysis, malware authors employ anti-debugging, anti-virtual machine, anti-sandbox techniques using sophisticated environment aware approaches [14, 15].

These obfuscation attacks can also be categorized into transformations which without modifying source code can thwart signature based static analysis and transformations which modify the source code files like classes.dex, AndroidManifest.xml or String.xml without changing the intended semantics of the application as given in Figure 5.

6.1.1 Anti-static techniques

Some of the anti-static techniques are given below:

- **Compression and Encryption:**
  These techniques encrypt and/or compress classes in the .dex file and place them in the data array. During runtime, a corresponding decompressor/decrypter will decrypt and/or decompress and load the malicious module in memory [16]. Packing programs [11] use one or more layers of compression on the sensitive payload of the malware executable. As a result, a static analyzer has access only to the packed file and the original content cannot be accessed directly. Crypters deploy simple encryption techniques like encrypting sensitive strings using XOR operations to confine access to static analyzers [11, 16].
- **Code and Data Obfuscation:**
  Malware authors use code transposition to reorder the code sequence while retaining the intended behavior. They utilize placement of unconditional branching statements, jump instructions and swapping of independent instructions. Binary footprint of the application can be altered by increasing the bytecode size. This can be done through the addition of Dalvik No-Operations (NOP) into arbitrary disassembled methods to alter the ordinance of the program code [17]. Data obfuscation involves trivial transformations like identifier renaming and modifying package names [16].
- **Anti-disassembly:**
  Malware authors write their malicious programs in high level languages and convert them into target binaries and distribute them. To analyze

the target binaries, malware analyst converts the machine code into interpretable form by reverse engineering tools like disassemblers [8]. A disassembler converts machine code into assembly level language. Malware authors target the way disassemblers work and take benefit of few assumptions which disassemblers usually make and tricks them to produce a false disassembly listing [11, 17].

- **Polymorphism:**
  Polymorphic malware incorporates multiple obfuscation techniques to generate innumerable variants to thwart signature matching. Techniques such as reassignment of registers, supplementing No-Operations (NOP), performing XOR operation and reordering of sub-routines are performed [16].
- **Android specific obfuscation techniques:**
  Some additional options at hand for malware authors specific to Android applications includes:
  - Repack: Repacking and realigning of android applications is possible without any bytecode changes to hinder signature based anti-malware [16, 17].
  - MultiDex: Applications split into multiple dex files can be used to hide the malware payload across multiple dex files and hinder detection [16, 17].
  - Native code: Native code are function calls invoked on external binaries like shared libraries. Malware misuse this functionality and embed malicious payload in external binaries which are invoked during run-time [16, 17].
  - Reflection: Reflection is the capability of an application to monitor itself at runtime to perform some bug fixes or upgrade to latest versions. Malware can utilize this capability for executing malicious content [11, 17].
  - Dynamic Code Loading: Malware can possibly extend their malicious functionality through dynamic loading of untrusted contents from remote websites by harnessing the Davik class loading capability [16].

6.1.2 Anti-dynamic techniques

Even though the signature is modified, the behavior of the generated malware remains identical. So utilization of behavioral analysis techniques like sandboxing, emulators can be used to detect malware [8]. But malware authors

have scaled up their skills and have developed analysis avoidance techniques encompassing anti-debugging, anti-emulation, anti-virtual machines and anti-sandboxing techniques [14].

- **Anti-Debugging:**
  Debuggers help to dissect the diverse tasks performed during execution of a running program and analyze its live memory content. Anti-debugging techniques investigating the presence of a debugger can serve as a hint to the malware that it is being analyzed [56].
- **Anti-Virtual machine and Anti-Emulator:**
  Malware analysts prefer to analyze malware in an emulated or virtual environment owing to the benefits such as the ability to roll back to a clean state after infection, re-observing the behavior of the malware through snapshots and concurrent execution of multiple samples. This also helps in analyzing malware without impacting any production workstations. To thwart the above analysis, malware authors utilize anti-virtual machine and anti-emulation techniques. These techniques check their current execution environment parameter values against possible physical environment parameter values and exhibit dormant behavior based on deviations observed [56].But, nowadays employing these techniques will be less effective as more potential real users utilize virtual machines with the increased usage of virtualization technology and cloud computing paradigms.
- **Anti-Sandbox:**
  Sandbox [11] enable the execution of malware in a safe and controlled scenario by making use of virtualization techniques. This behavioral analysis technique helps observe the behavior of the malware and provides meta-data like file, network and registry activities. Anti-sandbox evasive techniques deployed by malware authors include observing user interaction such as user mouse clicks and keyboard movements [56], fingerprinting virtual hardware, delaying execution and awareness of the environment. Awareness of the environment comprises observing deviations in a real physical environment against a sandboxed environment such as system properties. This includes system uptime, network traffic, cookies, system files, rebooting of the OS, checking total physical memory size and checking registry relevant artifacts.
  Evaluation frameworks for assessing anti-malware detection performance to various obfuscation techniques have been extensively analyzed in [16, 17, 158].

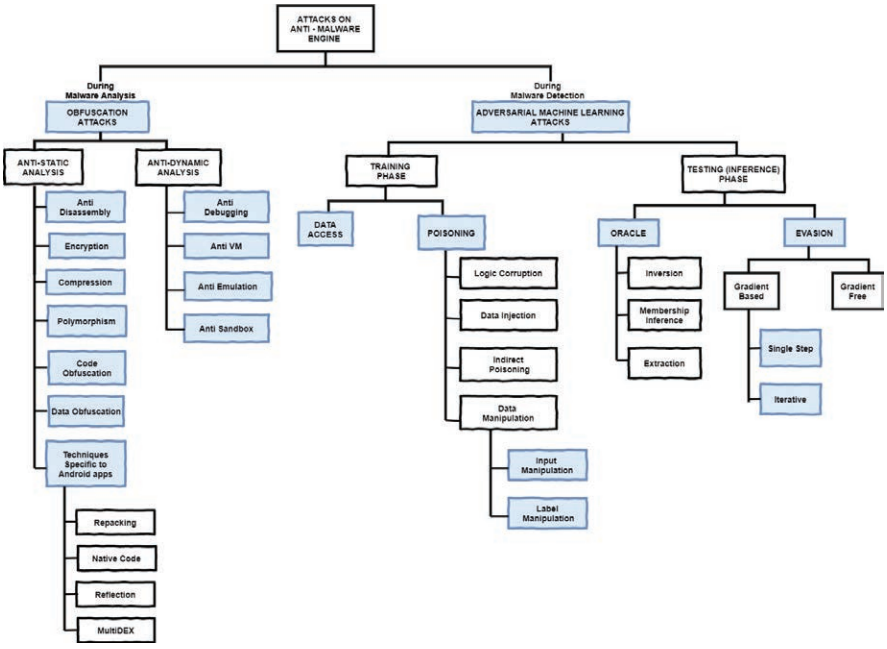

Figure 5 Attacks on anti-malware engine partially adapted from [36].

6.2 Adversarial Attacks – Disturbance caused during Malware Detection

Adversarial attacks are attacks launched on the learning based models incorporated in anti-malware engine to disrupt its working [31, 33, 159–164]. An adversarial attack is one which violates few statistical assumptions made by the ML models to some degree so as to confuse the model behavior. Gartner [165] reports the risks arising from adversarial attacks in its February 2019 report. Intelligent adversaries perform deliberate assaults on ML based systems to suppress the model from providing Confidentiality or Integrity or Availability, the CIA triad. The Integrity issues of learning based models [161] encountered in an adversarial environment utilized for malware detection scenario are to be analyzed in depth. Data Integrity and System Integrity [166] issues of learning based models questions the authenticity of the model's predictions against the real data generating distribution's outcome. Traditionally, most ML models were developed with a delicate threat model intended for non-malignant environments [24]. ML models inherently operate on the Independent and Identically Distributed (IID) assumption

[167]. IID assumption states the usage of the same data distribution to generate independent samples for training and testing phases. These assumptions are possible vulnerabilities which could be exploited by an attacker. Data scientists are usually more concerned about the efficiency of the model but they tend to miss on the security and robustness aspects of the model. ML models under adversarial environment is termed as Adversarial Machine Learning (AML) [161, 162]. AML attempts to hamper the learning models by crafting adversarial inputs. Adversaries attempt to cause the model to get confused and give erroneous outputs. AML [161] specifies techniques used by an intelligent adversary to alter the input samples with the intention to break a ML model when it is being trained or when it is making inferences. This creates room for various forms of attack models to compromise the security of the learning systems [7,31–33,48,161,162] [34–36,164,168–172].

The main aim of adversarial example crafting in malware detection [7, 48, 164, 168] is to misguide the malware detection system so as to change the detection for a given application as per the desire of the attacker. Therefore, the development of secure and robust ML models is necessary to protect against the adversarial attacks. Adversarial attack threat model helps us define the attack and its risk level. Attacks can be categorized based on various parameters like attack timing, attacker knowledge and attacker's goal [36, 161].

6.2.1 Attacker's knowledge

The knowledge possessed by the attacker can be based on his exposure to the various factors used for threat modeling like data used, features, learning model, architecture and model parameters. Based on the knowledge level of the attacker on the these factors, the attacks can be categorized as White Box attack for Perfect Knowledge (PK), Grey Box attack for Limited Knowledge (LK) and Black Box attack for Zero Knowledge (ZK) [161]. In Black box attack, the attacker has the base knowledge of whether static or dynamic analysis is performed to extract the features.

6.2.2 Attack timing

As per the NIST Taxonomy for AML, [36] the attacker can aim to disturb during the training phase or the inference phase as given in the Figure 5.

(a) Attacks during training phase

Attacks at training phase can be categorized as poisoning attacks and data access attacks.

(i) Poisoning attacks:

Poisoning attacks or causative attacks tend to intervene with the model building process or the training sample contents. This attack makes the model to produce correct outcomes for most of the inputs but yields wrong outcomes for few specific inputs. The different methodologies in poisoning attack are as follows:

- **Data injection attack:**
  Poisoning can be done by contaminating the dataset by the addition of spurious data. When this manipulated data samples are used for training, it results in data injection attack.
- **Data manipulation attack:**
  In this attack, the adversary performs transformation of existing samples which are of two subtypes:
  - Label manipulation attack:
    In this method, the label of a training sample is modified [173]. The adversary has access only to interfere into the training labels.
  - Input manipulation attack:
    The adversary is capable of altering the input feature values in addition to the labels of the training points as per the adversary requirement [174].
- **Logic corruption attack:**
  When an adversary alters the learning logic, a logic corruption attack is launched. Adversary makes sufficient alterations to interfere into the learning logic of the model thus manipulating the complete model.
- **Indirect poisoning attack:**
  The adversary [175] tries to poison the raw data. As direct access may not be available to the preprocessed features which are fed for training the model.

(ii) Data Access attacks:

In data access attack, the adversary gets partial or full access to the data used for training the model, using which a substitute model can be built. Adversary can use this surrogate model to later launch evasion attacks on this trained model [52].

(b) Attacks during Inference Phase:

Inference time attacks or Exploratory attacks [36, 161] which happens during the inference phase to disturb the trained model. It can be classified into Oracle and Evasion attacks.

(i) Oracle attacks:

Oracle attacks are similar to chosen plaintext attacks in cryptanalysis [176]. These attacks use the input-output pairs obtained from the model as the attacker does not have access to the intricacies of the model. Oracle attacks can be classified into extraction attack, inversion attack and membership inference attack.

- Extraction attack tries to extract the parameters of the model based on the input-output pairs derived from the model. With the obtained information about the internal structure of the model, the adversary will train and produce a substitute model with the exact behavior of the target model [177].
- Inversion attack tries to reconstruct user's private data causing violation of privacy [178].
- Membership Inference attack enquires whether an adversarial generated sample is part of the data distribution used for training [179].

(ii) Evasion attacks:

In evasion attacks [36, 180] the attacker crafts adversarial samples by perturbing the input samples which when fed to the trained model causes misclassification. Evasion attacks can be gradient based or gradient free.

- Gradient based attacks:
  In these attacks, we assume the adversary has access to the gradient of the model and utilizes the same to craft the adversarial samples. This includes single step attacks and iterative attacks.
  - In Single step attacks: The perturbations are generated by computing the gradient of the loss function only once. These attacks are also called as one shot attacks. An example method for one shot adversarial crafting is the Fast Gradient Sign Method (FGSM) [181].
    The adversarial sample is calculated as given in Equation (1):

$$x' = x + \in .sign(\nabla_x J(\theta, x, y) \tag{1}$$

    where,
    x represents the original malicious input.
    y represents the targets associated with x, the label as malware.
    x′ represents the adversarial sample

$$J(\theta, x, y)$$

represents the loss function of the model.
This method calculates the gradient of the loss function with respect to each input only once.

– Iterative attacks:
Computes the gradients iteratively and crafts the adversarial samples. The commonly used iterative attack based on the gradients for better sample generation is the Jacobian Saliency Map Attack (JSMA) [182]. JSMA computes the Jacobian matrix for the sample x using Equation (2).

$$J_F = \frac{\delta(F(x))}{x} = \left[\frac{\delta F_j(x)}{\delta x_i}\right]_{i*j} \tag{2}$$

Where, F denotes the logits layer of the model.
This matrix helps to find the input features which are contributing more to the target label. The model can be fooled by adding perturbations to the most significant features found using Jacobian matrix. This method iteratively finds the significant features until the maximum iterations are reached or the model incorrectly classifies the target label.

- Gradient free attacks access the model's confidence score to craft the adversarial samples [183].

6.2.3 Attacker's goal

Classifier's output integrity can be impacted by the choice of the attacker's goal. The goals can fit into confidence reduction, misclassification and source/target misclassification [162]. Confidence reduction is to reduce the end user's confidence on the model prediction. Model produces wrong output for all instances irrespective of benign or malware sample. Misclassification causes the effect similar to an untargeted attack where the model's output is different from its actual label. In malware scenario, the attacker's goal is generally to cause misclassification of the malware instance into benign instance. Source/target misclassification causes the output of a particular input instance to be classified as a specific class label.

Usually model training phase is inaccessible to malware authors as most of the commonly used anti-malware solutions are proprietary. This results in evasion attacks being common [32, 168]. Table 2 summarizes adversarial attack techniques applied in the malware domain. The abbreviations used in Table 2 are expanded in Table 3.

Table 2 Summary of adversarial attacking techniques for Malware

S.No	Target Dataset Used/ Targeted ML algorithm	Attack Methodology / [Ref]	Attack (P/E)/ Attacker's Knowledge (ZK/PK/LK)	Attack Properties
1	Android / SVM and Sec SVM	Problem space attack / [39]	E/ PK	Evades detection by code addition through automatic transplanting of benign code fragments while ensuring malicious functionality.
2	PDF / SVM and NN	[180] / GD attack	E / PK and LK	Theoretically works against any classifier with a differentiable discriminant function.
3	PDF/ SVM , RF	[184] / Genetic programming approach	E / LK	Searches for evasive variants using genetic programming. Idea is to stochastically manipulate a malicious sample to find a variant that preserves the malicious behavior but is classified as benign by the classifier.
4	Android / kNN, DT, SVM	[185] / Malware Evolution Attack and Malware Confusion Attack	E / LK	Proposes practical attacks that mutate malware variants to evade detection. Leverages existing malware program structures to change the features that are non-essential to malware but important to malware detectors.
5	Windows / GBDT	[42] / Reinforcement Learning (RL) based generic black-box attack	E / ZK	RL agents can automatically create novel Windows PE evasive malware variants by the modifying binary files.

6	PDF / RF.	[186] / GD-KDE attack (Mimicry)	E / PK	Exploits semantic gaps by injecting additional features that are not interpreted by PDF renderers.
7	Android / RF, KNN,SVM	[40] / CW and JSMA attack (Mimicry)	E / LK	Crafts adversarial example by either optimizing an adversarial objective function, or perturbing influential features based on the indicative forward derivatives.
8	Windows Dynamic analysis Sandox:MIST malware behavior reports. / Single (or complete) linkage hierarchical clustering .	[187] / Bridging attacks	P / PK	Reduces the attack to an optimization problem. The objective function to be maximized is the distance of the clusters formed when under attack to the clusters formed in absence of the attack. The attacker's goal is to cause clusters to merge until the system becomes unusable.
9	Windows API calls / SVM, DT, RF, LR, MLP and voting based ensemble of these classifiers (VOTE). .	[156] / MalGAN	E / ZK	Generates malware examples using a GAN. Game between Generator and Discriminator happens until equilibrium or fixed number of iterations.
10	Android / DNN	[164] / Uses JSMA	E / LK	Attack handles binary features while at the same time preserves the Android applications malicious functionality.
11	Windows/ Kaggle MS BIG 2015 dataset CNN	[43] / Iterative-FGSM attack	E / PK	Generates Adversarial Examples for discrete sequences by injecting local perturbations in an embedding space and reconstruct back to the discrete input space.

(*Continued*)

Table 2 Continued

S.No	Target Dataset Used/ Targeted ML algorithm	Attack Methodology / [Ref]	Attack (P/E)/ Attacker's Knowledge (ZK/PK/LK)	Attack Properties
12	Windows / MalConv	[44] / GD attack, Random Append, Gradient Append based attacks	E / PK	Appends adversarial noise to the end to the binary in the working of E2E CNN malware detector.
13	Windows / MalConv, E2E CNN malware detector	[45] / Benign FGM append, Slack FGM Attacks	E / PK	Infers that the MalConv architecture does not encode positional information about the input features and is vulnerable to append-based attacks.
14	Windows API call sequences run on Cuckoo Sandbox / RNN, FFNN, SVM	[41] / Mimicry attack based on FGSM using Jacobian matrix of the surrogate model	E / ZK	Implements GADGET, a software framework to convert any malware binary to a binary undetected by malware classifiers.
15	Windows API call sequences run on Cuckoo Sandbox / RNN variants, DNN, SVM, GBDT	[188] / Logarithmic backtracking, Random and Benign Perturbation to handle ML as a service (MLaaS) attacks	E / ZK	Attack uses a sliding window approach: Each API call sequence is divided into windows with size 'm'. Detection is performed on each window in turn, if any window is classified as malicious, entire sequence is considered malicious.

Table 3 Summary of abbreviations used in Table 2

E – Evasion	FGSM – Fast Gradient Sign Method	RF – Random Forest
P – Poisoning	JSMA – Jacobian Saliency Map Attack	kNN – k-Nearest Neighbors
PK – Perfect Knowledge	SVM – Support Vector Machine	LR – Logistic Regression
LK – Limited Knowledge	NN – Neural Network	CNN – Convolutional NN
ZK – Zero Knowledge	DNN – Deep NN	FFNN – Feed Forward NN
CW – Carlini-Wagner	MLP – Multi Layer Perceptron	RNN – Recurrent NN
GD – Gradient Descent	DT – Decision Tree	GBDT – Gradient Boosted DT

7 Defense Modelling

Defense methods to handle adversarial attacks against learning based malware detectors can be modelled as active or passive techniques depending upon whether they are actively identifying the adversarial samples or passively increasing the robustness of the underlying classifier to such attacks. They aim to improve the robustness of the model to provide a secure learning model. Defense methods deployed can be build against specific attacks or can be attack agnostic which [58, 164, 168, 185, 189] be the preferred case. Defense methodologies [7, 46, 48, 49, 190, 191] can be roughly summarized as but not limited to the following:

- Monotonic classification techniques hardens the model by increasing the cost of deceiving the classifier. The classifier is trained by ordaining monotonicity on particular selected features which improves the robustness of the classifier [46].
- Security by obscurity relies on stealth for protection like defensive distillation [192].
- Perimeter defense model such as adversarial retraining does retraining of the classifier with adversarial samples included with correct labels [181, 182, 193]
- Defense in depth model use robust approaches like data sanitization methods, (Reject on Negative Impact RONI [194]), generative models (defense GAN [170], denoising techniques (Magnet [195], high-level representation guided denoiser [172]), feature squeezing techniques

Table 4 Summary of adversarial defense techniques for malware

S.No	Dataset Used / [Ref]Learner/ Defense Against (P/E)	Defense Properties
1	Windows/ [46] / XGBoost/ Evasion attack	The classifier is trained by ordaining monotonicity on particular selected features which improves the robustness of the classifier. Monotonic classification hardens the model by increasing the cost of deceiving the classifier.
2	Android / [48]/ FARM/ Increased fake API package call attack, increased percentage of permissions attack and reduced percentage of API package call attack	Applies irreversible transformations such as landmark based, feature clustering based and correlation graph based transformations to transforms the extracted features into new feature space which improves robustness to attacks.
3	Windows/ [199]/ SecDefender/ E	Performs feature manipulations to enhance the robustness against attacks.
4	Android (Drebin, Contagio) / [200]/ Sec-SVM/ Obfuscation attacks, and evasion attacks with feature addition and feature removal	Defines a novel, theoretically-sound learning algorithm to train linear classifiers with more evenly-distributed feature weights.
5	Android / [49]/ Count featurization technique for feature space transformation/ AdvAttack,(E)	Implements count featurization for feature manipulations available to an adversarial attack, and introduces softmax function with adversarial parameter.
6	PDF / [201]/ Wrapper-Based Adversarial Feature Selection (WAFS) SVM with the RBF kernel/ Gradient-descent based evasion attacks	Provides robustness to evasion attacks using forward feature selection and backward feature elimination algorithms.
7	Android / [190]/ SecCLS (Feature Selection Method) SecENS (Ensemble Learner)/ Feature Manipulation using brute-force attacks	Enforces attackers to manipulate a large number of features

8	Windows/ [202]/ Ensemble of NN based classifiers/ Gradient descent based adversarial training	Systematizes few principles for enhancing the robustness of neural network classifiers.
9	Android/ [50]/ DNN/ JSMA, GD-KDE, CW attack, Mimicry attack	Uses hash functions with a certain locality-preserving property to transform samples to enhance the robustness of DNNs. Uses a Denoising Auto-Encoder (DAE) regularizer.
10	Windows/ [203]/ Weight decay defense DNN/ JSMA	Demonstrates that adding additional hidden layers to NN models does significantly increase the classifier's robustness to adversarial attacks.
11	Windows/ [51]/ Proposed SLEIPNIR framework/ FGSM, Multi-Step Bit Gradient Ascent, Multi-Step Bit Coordinate Ascent attack	Investigates methods that reduce the adversarial blind spots for NN malware detectors. This is approached as a saddle-point optimization problem.
12	PDF / [204]/ Evidence-based ML threat modeling/ EvadeML	Suggests usage of conserved features which are constrained to remain unmodified in the presence of evasion attack.

which offer model hardening [196] and ensemble based approaches [197]
- Game theoretic techniques [169] can be used which are theoretically sound but are computationally demanding and has scalability issues to large data.
- Utilize the data transformation techniques which could block the backward flow of gradients. Such data transformations must satisfy following three requirements: Non-Differentiable, computationally intractable and preserve the distribution of input representation [198].
- Utilize robust feature transformation method like FARM [48]

Table 4 summarizes adversarial defense methods applied in the malware domain. Performance indicators utilized specifically for AML are evasion rate and distortion rate [182].

Evasion rate is used for measuring how vulnerable a learner is. It is defined as the percentage of malware samples that are classified as benign after being altered but were correctly classified before. Distortion rate is the average number of modifications required to modify the malware sample to achieve misclassification.

Most of the existing works on adversarial learning are in the context of images. Images are continuous space where perturbations can be crafted anywhere in the image to cause misclassification. In the malware space, the attacker faces few constraints when crafting the adversarial samples [7, 48, 164, 168].

- Attack has to deal with discrete binary space instead of continuous space.
- Adversarial attacks have to be crafted using the good word attack approach [205], or follow the Mimicry attack [206]. Features can be only added to the malware sample and existing features cannot be removed as it may change the semantics of the malware's intended functionality.
- The cost of creating the adversarial sample should be lesser than the value gained by performing the attack.

8 Conclusion and Future Work

Digitization has brought about vast improvements and increased sophistication for users of technology. Simultaneously, however, the development of technology has also subjected users to a higher level of risk in security and privacy infringement. This review article provides an in-depth study on

existing Android malware and techniques for malware analysis and detection. It also elaborates on the challenges faced in using the existing tools and techniques. These limitations put forward the need to untangle -the difficulties in development of an anti-malware engine recommending an adversary aware pro-active approach.Hence, it is necessary to build a robust obfuscation resilient malware detector. This can be achieved by devising techniques and models to answer the following queries:

- How to collect data with a balanced set of samples comprising malware and benign samples. Within the malware samples, how to compile samples from the diversified malware families available?
- How to combine static and dynamic analysis for effective and efficient feature retrieval?
- How to extract features? What are the tradeoffs between lightweight versus heavyweight feature extraction techniques?
- Which feature selection techniques to use? What is the threshold on the number of features to use?
- How to anticipate and handle obfuscated attacks?
- Which learning technique to be deployed? What is the choice of the classifier for binary and multi-class classification? The detection of the family of a malware app can be coarse grained (e.g., Trojan,virus, worm, spyware, etc.) or finer grained(e.g., Droid-KungFu, DroidDream, Oldboot, etc.) .
- How vulnerable is the learner to the prevalent adversarial attacks?
- How to safeguard the learner and harden its defense against the adversarial attacks by incorporating the principle of security by design approach?
- How to evaluate the performance of the model from the perspective of generalization and robustness against attacks?
- How to develop on-device malware detection application for real-time scanning of malware?

In short, the discussions briefed in this technical review can foster research in Android malware detection.

References

[1] W. Han, J. Xue, Y. Wang, L. Huang, Z. Kong, and L. Mao, "Maldae: Detecting and explaining malware based on correlation and fusion of static and dynamic characteristics," *Computers & Security*, vol. 83, pp. 208–233, 2019.

[2] J.-Y. Kim, S.-J. Bu, and S.-B. Cho, "Zero-day malware detection using transferred generative adversarial networks based on deep autoencoders," *Information Sciences*, vol. 460, pp. 83–102, 2018.
[3] S. Y. Yerima and S. Sezer, "Droidfusion: A novel multilevel classifier fusion approach for android malware detection," *IEEE transactions on cybernetics*, vol. 49, no. 2, pp. 453–466, 2018.
[4] W. Yuan, Y. Jiang, H. Li, and M. Cai, "A lightweight on-device detection method for android malware," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 2019.
[5] C. Hasegawa and H. Iyatomi, "One-dimensional convolutional neural networks for android malware detection," in *2018 IEEE 14th International Colloquium on Signal Processing & Its Applications (CSPA)*, pp. 99–102, IEEE, 2018.
[6] Z.-U. Rehman, S. N. Khan, K. Muhammad, J. W. Lee, Z. Lv, S. W. Baik, P. A. Shah, K. Awan, and I. Mehmood, "Machine learning-assisted signature and heuristic-based detection of malwares in android devices," *Computers & Electrical Engineering*, vol. 69, pp. 828–841, 2018.
[7] C. Bai, Q. Han, G. Mezzour, F. Pierazzi, and V. Subrahmanian, "Dbank: Predictive behavioral analysis of recent android banking trojans," *IEEE Transactions on Dependable and Secure Computing*, 2019.
[8] M. Christodorescu and S. Jha, "Static analysis of executables to detect malicious patterns," tech. rep., WISCONSIN UNIV-MADISON DEPT OF COMPUTER SCIENCES, 2006.
[9] A. Moser, C. Kruegel, and E. Kirda, "Limits of static analysis for malware detection," in *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pp. 421–430, IEEE, 2007.
[10] I. You and K. Yim, "Malware obfuscation techniques: A brief survey," in *2010 International conference on broadband, wireless computing, communication and applications*, pp. 297–300, IEEE, 2010.
[11] M. Sikorski and A. Honig, *Practical malware analysis: the hands-on guide to dissecting malicious software*. no starch press, 2012.
[12] A. Aghamohammadi and F. Faghih, "Lightweight versus obfuscation-resilient malware detection in android applications," *Journal of Computer Virology and Hacking Techniques*, pp. 1–15, 2019.
[13] J. Garcia, M. Hammad, and S. Malek, "Lightweight, obfuscation-resilient detection and family identification of android malware," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 26, no. 3, pp. 1–29, 2018.

[14] S. Sen, E. Aydogan, and A. I. Aysan, "Coevolution of mobile malware and anti-malware," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 10, pp. 2563–2574, 2018.

[15] M. Brand, C. Valli, and A. Woodward, "Malware forensics: Discovery of the intent of deception," *Journal of Digital Forensics, Security and Law*, vol. 5, no. 4, p. 2, 2010.

[16] V. Rastogi, Y. Chen, and X. Jiang, "Catch me if you can: Evaluating android anti-malware against transformation attacks," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 1, pp. 99–108, 2013.

[17] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, "Stealth attacks: An extended insight into the obfuscation effects on android malware," *Computers & Security*, vol. 51, pp. 16–31, 2015.

[18] D. Gibert, C. Mateu, and J. Planes, "The rise of machine learning for detection and classification of malware: Research developments, trends and challenges," *Journal of Network and Computer Applications*, p. 102526, 2020.

[19] P. Faruki, A. Bharmal, V. Laxmi, V. Ganmoor, M. S. Gaur, M. Conti, and M. Rajarajan, "Android security: a survey of issues, malware penetration, and defenses," *IEEE communications surveys & tutorials*, vol. 17, no. 2, pp. 998–1022, 2014.

[20] Mcafee, "evolution-of-malware-sandbox-evasion-tactics-a-retrospective-study." https://www.mcafee.com/blogs/other-blogs/mcafee-labs/evolution-of-malware-sandbox-evasion-tactics-a-retrospective-study/, 2019.

[21] Sophos. https://www.sophos.com/en-us/press-office/press-releases/2017/11/sophos-adds-deep-learning-capabilities-to-intercept-x-early-access-program.aspx, 2019.

[22] Cylance. https://www.cylance.com/en-us/resources/knowledge-center/ai-and-ml-for-security.html, 2019.

[23] CrowdStrike. https://www.crowdstrike.com/blog/defending-against-malware-with-machine-learning/, 2019.

[24] cleverhans. http://www.cleverhans.io/security/privacy/ml/2016/12/16/breaking-things-is-easy.html, 2019.

[25] darpa.mil. https://www.darpa.mil/news-events/2019-02-06, 2019.

[26] ai.google. https://ai.google/responsibilities/responsible-ai-practices/, 2019.

[27] adversarial robustness. https://github.com/IBM/adversarial-robustness-toolbox/, 2019.

[28] securing-artificial intelligence. https://docs.microsoft.com/en-us/security/engineering/securing-artificial-intelligence-machine-learning, 2019.
[29] secml.gitlab. https://secml.gitlab.io/, 2019.
[30] tensorflow cleverhans. https://github.com/tensorflow/cleverhans, 2019.
[31] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, and D. Mukhopadhyay, "Adversarial attacks and defences: A survey," *arXiv preprint arXiv:1810.00069*, 2018.
[32] S. Qiu, Q. Liu, S. Zhou, and C. Wu, "Review of artificial intelligence adversarial attack and defense technologies," *Applied Sciences*, vol. 9, no. 5, p. 909, 2019.
[33] X. Yuan, P. He, Q. Zhu, and X. Li, "Adversarial examples: Attacks and defenses for deep learning," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 9, pp. 2805–2824, 2019.
[34] I. Goodfellow, "Defense against the dark arts: An overview of adversarial example security research and future research directions," *arXiv preprint arXiv:1806.04169*, 2018.
[35] I. Goodfellow, P. McDaniel, and N. Papernot, "Making machine learning robust against adversarial inputs," *Communications of the ACM*, vol. 61, no. 7, pp. 56–66, 2018.
[36] E. Tabassi, K. J. Burns, M. Hadjimichael, A. D. Molina-Markham, and J. T. Sexton, "A taxonomy and terminology of adversarial machine learning," 2019.
[37] medialibrary. https://www.sophos.com/en-us/medialibrary/pdfs/technical-papers/sophoslabs-uncut-2020-threat-report.pdf, 2019.
[38] skylightcyber. https://skylightcyber.com/2019/07/18/cylance-i-kill-you/, 2019.
[39] F. Pierazzi, F. Pendlebury, J. Cortellazzi, and L. Cavallaro, "Intriguing properties of adversarial ml attacks in the problem space," in *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 1308–1325, IEEE Computer Society, 2020.
[40] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android hiv: A study of repackaging malware for evading machine-learning detection," *arXiv preprint arXiv:1808.04218*, 2018.
[41] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Generic black-box end-to-end attack against state of the art api call based malware classifiers," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 490–510, Springer, 2018.

[42] H. S. Anderson, A. Kharkar, B. Filar, D. Evans, and P. Roth, "Learning to evade static pe machine learning malware models via reinforcement learning," *arXiv preprint arXiv:1801.08917*, 2018.

[43] F. Kreuk, A. Barak, S. Aviv-Reuven, M. Baruch, B. Pinkas, and J. Keshet, "Deceiving end-to-end deep learning malware detectors using adversarial examples," *arXiv preprint arXiv:1802.04528*, 2018.

[44] B. Kolosnjaji, A. Demontis, B. Biggio, D. Maiorca, G. Giacinto, C. Eckert, and F. Roli, "Adversarial malware binaries: Evading deep learning for malware detection in executables," *arXiv preprint arXiv:1803.04173*, 2018.

[45] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," *arXiv preprint arXiv:1810.08280*, 2018.

[46] Í. Íncer Romeo, M. Theodorides, S. Afroz, and D. Wagner, "Adversarially robust malware detection using monotonic classification," in *Proceedings of the Fourth ACM International Workshop on Security and Privacy Analytics*, pp. 54–63, 2018.

[47] I. Rosenberg, A. Shabtai, Y. Elovici, and L. Rokach, "Defense methods against adversarial examples for recurrent neural networks," *arXiv preprint arXiv:1901.09963*, 2019.

[48] Q. Han, V. Subrahmanian, and Y. Xiong, "Android malware detection via (somewhat) robust irreversible feature transformations," *IEEE Transactions on Information Forensics and Security*, 2020.

[49] L. Chen, S. Hou, Y. Ye, and S. Xu, "Droideye: Fortifying security of learning-based classifier against adversarial android malware attacks," in *2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pp. 782–789, IEEE, 2018.

[50] D. Li, R. Baral, T. Li, H. Wang, Q. Li, and S. Xu, "Hashtran-dnn: A framework for enhancing robustness of deep neural networks against adversarial malware samples," *arXiv preprint arXiv:1809.06498*, 2018.

[51] A. Al-Dujaili, A. Huang, E. Hemberg, and U.-M. O'Reilly, "Adversarial deep learning for robust detection of binary encoded malware," *arXiv preprint arXiv:1801.02950*, 2018.

[52] S. Chen, M. Xue, L. Fan, S. Hao, L. Xu, H. Zhu, and B. Li, "Automated poisoning attacks and defenses in malware detection systems: An adversarial machine learning approach," *computers & security*, vol. 73, pp. 326–344, 2018.

[53] D. Ucci, L. Aniello, and R. Baldoni, "Survey of machine learning techniques for malware analysis," *Computers & Security*, vol. 81, pp. 123–147, 2019.

[54] A. Qamar, A. Karim, and V. Chang, "Mobile malware attacks: Review, taxonomy & future directions," *Future Generation Computer Systems*, vol. 97, pp. 887–909, 2019.
[55] O. Or Meir, N. Nissim, Y. Elovici, and L. Rokach, "Dynamic malware analysis in the modern era—a state of the art survey," *ACM Computing Surveys (CSUR)*, vol. 52, no. 5, pp. 1–48, 2019.
[56] A. Afianian, S. Niksefat, B. Sadeghiyan, and D. Baptiste, "Malware dynamic analysis evasion techniques: A survey," *arXiv preprint arXiv:1811.01190*, 2018.
[57] C. S. Veerappan, P. L. K. Keong, Z. Tang, and F. Tan, "Taxonomy on malware evasion countermeasures techniques," in *2018 IEEE 4th World Forum on Internet of Things (WF-IoT)*, pp. 558–563, IEEE, 2018.
[58] D. Li, Q. Li, Y. Ye, and S. Xu, "Enhancing deep neural networks against adversarial malware examples," *arXiv preprint arXiv:2004.07919*, 2020.
[59] venturebeat. https://venturebeat.com/2019/05/07/android-passes-2-5-billion-monthly-active-devices/, 2019.
[60] cisomag. https://www.cisomag.com/mcafee-report-predicts-2020-to-be-year-of-mobile-sneak-attacks/, 2019.
[61] D. E. Denning, "An intrusion-detection model," *IEEE Transactions on software engineering*, vol. 1, no. 2, pp. 222–232, 1987.
[62] R. Bace and P. Mell, "Nist special publication on intrusion detection systems," tech. rep., BOOZ-ALLEN AND HAMILTON INC MCLEAN VA, 2001.
[63] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," tech. rep., National Institute of Standards and Technology, 2012.
[64] H.-J. Liao, C.-H. R. Lin, Y.-C. Lin, and K.-Y. Tung, "Intrusion detection system: A comprehensive review," *Journal of Network and Computer Applications*, vol. 36, no. 1, pp. 16–24, 2013.
[65] A. Torkaman, G. Javadzadeh, and M. Bahrololum, "A hybrid intelligent hids model using two-layer genetic algorithm and neural network," in *The 5th Conference on Information and Knowledge Technology*, pp. 92–96, IEEE, 2013.
[66] R. Puzis, M. D. Klippel, Y. Elovici, and S. Dolev, "Optimization of nids placement for protection of intercommunicating critical infrastructures," in *European Conference on Intelligence and Security Informatics*, pp. 191–203, Springer, 2008.

[67] B. Randell, *On Alan Turing and the origins of digital computers*. University of Newcastle upon Tyne, Computing Laboratory, 1972.
[68] J. G. Kemeny, "Theory of self-reproducing automata. john von neumann. edited by arthur w. burks. university of illinois press, urbana, 1966. 408 pp., illus. 10," 1967.
[69] Creeper. https://www.wikipedia.org/wiki/Creeper-(program), 2019.
[70] F. Cohen, *Computer viruses*. PhD thesis, University of Southern California Doctoral dissertation, 1986.
[71] F. Cohen, "Computer viruses: theory and experiments," *Computers & security*, vol. 6, no. 1, pp. 22–35, 1987.
[72] E. Messmer, "Tech talk: Where'd it come from, anyway?," *Pc World: Business Cen*, 2008.
[73] retaildive. https://www.retaildive.com/ex/mobilecommercedaily/a-brief-history-of-mobile-malware, 2019.
[74] malware. https://www.wikipedia.org/wiki/Mobile-malware, 2019.
[75] cybersecurity.att. https://cybersecurity.att.com/blogs/labs-research/analysis-of-trojan-sms.androidos.fakeplayer.a, 2019.
[76] J. E. Canavan, *Fundamentals of network security*. Artech House, 2001.
[77] S. Bahtiyar, "Anatomy of targeted attacks with smart malware," *Security and Communication Networks*, vol. 9, no. 18, pp. 6215–6226, 2016.
[78] M. La Polla, F. Martinelli, and D. Sgandurra, "A survey on security for mobile devices," *IEEE communications surveys & tutorials*, vol. 15, no. 1, pp. 446–471, 2012.
[79] f secure. https://www.f-secure.com/v-descs/brador.shtml, 2019.
[80] welivesecurity. https://www.welivesecurity.com/2019/07/29/android-ransomware-back/, 2019.
[81] blog.trendmicro. http://blog.trendmicro.com/trendlabs-security-intelligence/hackingteam-rcsandroid-spying-tool-listens-to-calls-roots-devices-to-get-in/, 2019.
[82] Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, and G. Vigna, "Triggerscope: Towards detecting logic bombs in android applications," in *2016 IEEE symposium on security and privacy (SP)*, pp. 377–396, IEEE, 2016.
[83] G. Hoglund and J. Butler, *Rootkits: subverting the Windows kernel*. Addison-Wesley Professional, 2006.
[84] Google. https://www.wikipedia.org/wiki/Google-Play, 2019.
[85] blog.checkpoint. https://blog.checkpoint.com/2016/07/01/from-hummingbad-to-worse-new-in-depth-details-and-analysis-of-the-humming bad-andriod-malware-campaign/, 2019.

[86] bleepingcomputer. https://www.bleepingcomputer.com/news/security/new-mysterybot-android-malware-packs-a-banking-trojan-keylogger-and-ransomware/, 2019.
[87] zdnet. https://www.zdnet.com/article/the-ultimate-guide-to-finding-and-killing-spyware-and-stalkerware/, 2019.
[88] B. C. Brown, *How to Stop E-mail Spam, Spyware, Malware, Computer Viruses, and Hackers from Ruining Your Computer Or Network: The Complete Guide for Your Home and Work*. Atlantic Publishing Company, 2010.
[89] Q. Hu and T. Dinev, "Is spyware an internet nuisance or public menace?," *Communications of the ACM*, vol. 48, no. 8, pp. 61–66, 2005.
[90] gadgets.ndtv. https://gadgets.ndtv.com/apps/news/17-android-apps-google-play-store-adware-discovered-bitdefender-2164911, 2019.
[91] nakedsecurity.sophos. https://nakedsecurity.sophos.com/2013/05/31/android-malware-in-pictures-a-blow-by-blow-account-of-mobile-scareware/, 2019.
[92] X. Luo and Q. Liao, "Awareness education as the key to ransomware prevention," *Information Systems Security*, vol. 16, no. 4, pp. 195–202, 2007.
[93] economictimes.indiatimes. https://economictimes.indiatimes.com/tech/internet/cognizant-hit-by-maze-ransomware-attack/articleshow/75228505.cms, 2019.
[94] tripwire. https://www.tripwire.com/state-of-security/security-data-protection/3-malware-trends/, 2019.
[95] itproportal. https://www.itproportal.com/features/new-types-of-android-ransomware/, 2019.
[96] A. Pastor, A. Mozo, S. Vakaruk, D. Canavese, D. R. López, L. Regano, S. Gómez-Canaval, and A. Lioy, "Detection of encrypted cryptomining malware connections with machine and deep learning," *IEEE Access*, vol. 8, pp. 158036–158055, 2020.
[97] techradar. https://www.techradar.com/in/news/android-banking-botnet-targets-thousands, 2019.
[98] wired. https://www.wired.com/story/google-android-chamois-botnet/, 2019.
[99] first-twitter controlled. https://www.welivesecurity.com/2016/08/24/first-twitter-controlled-android-botnet-discovered/, 2019.
[100] S. Kumar, "An emerging threat fileless malware: a survey and research challenges," *Cybersecurity*, vol. 3, no. 1, pp. 1–12, 2020.

[101] S. Mansfield-Devine, "Fileless attacks: compromising targets without malware," *Network Security*, vol. 2017, no. 4, pp. 7–11, 2017.
[102] security awareness. https://www.tripwire.com/state-of-security/security-awareness/fileless-malware-stop/, 2019.
[103] security technology. https://www.trendmicro.com/vinfo/in/security/news/security-technology/how-can-advanced-sandboxing-techniques-thwart-elusive-malware, 2019.
[104] comparitech. https://www.comparitech.com/blog/information-security/fileless-malware-attacks/, 2019.
[105] A. Balakrishnan and C. Schulze, "Code obfuscation literature survey," *CS701 Construction of compilers*, vol. 19, 2005.
[106] M. Lindorfer, M. Neugschwandtner, and C. Platzer, "Marvin: Efficient and comprehensive mobile app classification through static and dynamic analysis," in *2015 IEEE 39th annual computer software and applications conference*, vol. 2, pp. 422–433, IEEE, 2015.
[107] M. Christodorescu and S. Jha, "Testing malware detectors," *ACM SIGSOFT Software Engineering Notes*, vol. 29, no. 4, pp. 34–44, 2004.
[108] developer.android. https://developer.android.com/guide/components/fundamentals, 2019.
[109] H. Wang, J. Si, H. Li, and Y. Guo, "Rmvdroid: towards a reliable android malware dataset with app metadata," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 404–408, IEEE, 2019.
[110] K. Allix, T. F. Bissyandé, J. Klein, and Y. Le Traon, "Androzoo: Collecting millions of android apps for the research community," in *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pp. 468–471, IEEE, 2016.
[111] f droid.org. https://f-droid.org/en/, 2019.
[112] appsapk. http://www.appsapk.com/, 2019.
[113] apkpure. https://apkpure.com/., 2019.
[114] L. Taheri, A. F. A. Kadir, and A. H. Lashkari, "Extensible android malware detection and family classification using network-flows and api-calls," in *2019 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–8, IEEE, 2019.
[115] Y. Li, J. Jang, X. Hu, and X. Ou, "Android malware clustering through malicious payload mining," in *International Symposium on Research in Attacks, Intrusions, and Defenses*, pp. 192–214, Springer, 2017.
[116] A. H. Lashkari, A. F. A. Kadir, L. Taheri, and A. A. Ghorbani, "Toward developing a systematic approach to generate benchmark android

malware datasets and classification," in *2018 International Carnahan Conference on Security Technology (ICCST)*, pp. 1–7, IEEE, 2018.
[117] A. H. Lashkari, A. F. A. Kadir, H. Gonzalez, K. F. Mbah, and A. A. Ghorbani, "Towards a network-based framework for android malware detection and characterization," in *2017 15th Annual Conference on Privacy, Security and Trust (PST)*, pp. 233–23309, IEEE, 2017.
[118] A. F. A. Kadir, N. Stakhanova, and A. A. Ghorbani, "Android botnets: What urls are telling us," in *International Conference on Network and System Security*, pp. 78–91, Springer, 2015.
[119] D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, and C. Siemens, "Drebin: Effective and explainable detection of android malware in your pocket.," in *Ndss*, vol. 14, pp. 23–26, 2014.
[120] contagiominidump. http://contagiominidump.blogspot.com/s, 2019.
[121] code. https://code.google.com/p/androguard/, 2019.
[122] ibotpeaches. https://ibotpeaches.github.io/Apktool/, 2019.
[123] code.google. https://code.google.com/p/dex2jar, 2019.
[124] radare. http://radare.org/y/?p=download, 2019.
[125] dexter. http://dexter.dexlabs.org/., 2019.
[126] hex rays. https://www.hex-rays.com/wp-content/uploads/2019/12/debugging-dalvik.pdf, 2019.
[127] remnux. https://remnux.org/, 2019.
[128] redmine. http://redmine.honeynet.org/projects/are/wiki, 2019.
[129] developer. http://developer.android.com/tools/help/proguard.html, 2019.
[130] saikoa. http://www.saikoa.com/dexguard, 2019.
[131] dexprotector. http://dexprotector.com/, 2019.
[132] W. Wang, X. Wang, D. Feng, J. Liu, Z. Han, and X. Zhang, "Exploring permission-induced risk in android applications for malicious application detection," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1869–1882, 2014.
[133] X. Liu and J. Liu, "A two-layered permission-based android malware detection scheme," in *2014 2nd IEEE International Conference on Mobile Cloud Computing, Services, and Engineering*, pp. 142–148, IEEE, 2014.
[134] D.-J. Wu, C.-H. Mao, T.-E. Wei, H.-M. Lee, and K.-P. Wu, "Droidmat: Android malware detection through manifest and api calls tracing," in *2012 Seventh Asia Joint Conference on Information Security*, pp. 62–69, IEEE, 2012.

[135] Y. Shao, X. Luo, C. Qian, P. Zhu, and L. Zhang, "Towards a scalable resource-driven approach for detecting repackaged android applications," in *Proceedings of the 30th Annual Computer Security Applications Conference*, pp. 56–65, 2014.

[136] J. Song, C. Han, K. Wang, J. Zhao, R. Ranjan, and L. Wang, "An integrated static detection and analysis framework for android," *Pervasive and Mobile Computing*, vol. 32, pp. 15–25, 2016.

[137] A. Feizollah, N. B. Anuar, R. Salleh, G. Suarez-Tangil, and S. Furnell, "Androdialysis: Analysis of android intent effectiveness in malware detection," *computers & security*, vol. 65, pp. 121–134, 2017.

[138] F. Idrees, M. Rajarajan, M. Conti, T. M. Chen, and Y. Rahulamathavan, "Pindroid: A novel android malware detection system using ensemble learning methods," *Computers & Security*, vol. 68, pp. 36–46, 2017.

[139] D. Maiorca, F. Mercaldo, G. Giacinto, C. A. Visaggio, and F. Martinelli, "R-packdroid: Api package-based characterization and detection of mobile ransomware," in *Proceedings of the symposium on applied computing*, pp. 1718–1723, 2017.

[140] G. Meng, Y. Xue, Z. Xu, Y. Liu, J. Zhang, and A. Narayanan, "Semantic modelling of android malware for effective malware comprehension, detection, and classification," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, pp. 306–317, 2016.

[141] K. Zhao, D. Zhang, X. Su, and W. Li, "Fest: A feature extraction and selection tool for android malware detection," in *2015 IEEE symposium on computers and communication (ISCC)*, pp. 714–720, IEEE, 2015.

[142] S. Hahn, M. Protsenko, and T. Müller, "Comparative evaluation of machine learning-based malware detection on android.," *Sicherheit 2016-Sicherheit, Schutz und Zuverlässigkeit*, 2016.

[143] M. R. Amin, M. Zaman, M. S. Hossain, and M. Atiquzzaman, "Behavioral malware detection approaches for android," in *2016 IEEE International Conference on Communications (ICC)*, pp. 1–6, IEEE, 2016.

[144] G. Canfora, A. De Lorenzo, E. Medvet, F. Mercaldo, and C. A. Visaggio, "Effectiveness of opcode ngrams for detection of multi family android malware," in *2015 10th International Conference on Availability, Reliability and Security*, pp. 333–340, IEEE, 2015.

[145] D. Su, W. Wang, X. Wang, and J. Liu, "Anomadroid: Profiling android applications' behaviors for identifying unknown malapps," in *2016 IEEE Trustcom/BigDataSE/ISPA*, pp. 691–698, IEEE, 2016.

[146] S. Bhandari, R. Gupta, V. Laxmi, M. S. Gaur, A. Zemmari, and M. Anikeev, "Draco: Droid analyst combo an android malware analysis framework," in *Proceedings of the 8th International Conference on Security of Information and Networks*, pp. 283–289, 2015.

[147] K.-H.-T. Dam and T. Touili, "Learning android malware," in *Proceedings of the 12th International Conference on Availability, Reliability and Security*, pp. 1–9, 2017.

[148] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, "Semantics-aware android malware classification using weighted contextual api dependency graphs," in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pp. 1105–1116, 2014.

[149] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, "Hey you get off of my market: detecting malicious apps in official and alternative android markets," in *NDSS*, vol. 25, pp. 50–52, 2012.

[150] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets," in *Proceedings of the 36th International Conference on Software Engineering*, pp. 175–186, 2014.

[151] T. Chen, Q. Mao, Y. Yang, M. Lv, and J. Zhu, "Tinydroid: a lightweight and efficient model for android malware detection and classification," *Mobile Information Systems*, vol. 2018, 2018.

[152] G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, and L. Cavallaro, "Droidsieve: Fast and accurate classification of obfuscated android malware," in *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pp. 309–320, 2017.

[153] H. Gonzalez, N. Stakhanova, and A. A. Ghorbani, "Droidkin: Lightweight detection of android apps similarity," in *International Conference on Security and Privacy in Communication Networks*, pp. 436–453, Springer, 2014.

[154] M. Ahmadi, A. Sotgiu, and G. Giacinto, "Intelliav: Building an effective on-device android malware detector," *arXiv preprint arXiv:1802.01185*, 2018.

[155] N. Islam, S. Das, and Y. Chen, "On-device mobile phone security exploits machine learning," *IEEE Pervasive Computing*, vol. 16, no. 2, pp. 92–96, 2017.

[156] W. Hu and Y. Tan, "Generating adversarial malware examples for black-box attacks based on gan," *arXiv preprint arXiv:1702.05983*, 2017.

[157] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," 1997.

[158] P. Graux, J.-F. Lalande, and V. V. T. Tong, "Obfuscated android application development," in *Proceedings of the Third Central European Cybersecurity Conference*, pp. 1–6, 2019.

[159] D. Lowd and C. Meek, "Adversarial learning," in *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pp. 641–647, 2005.

[160] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, pp. 43–58, 2011.

[161] N. Papernot, P. McDaniel, A. Sinha, and M. Wellman, "Towards the science of security and privacy in machine learning," *arXiv preprint arXiv:1611.03814*, 2016.

[162] N. Papernot, P. McDaniel, S. Jha, M. Fredrikson, Z. B. Celik, and A. Swami, "The limitations of deep learning in adversarial settings," in *2016 IEEE European symposium on security and privacy (Euro S and P)*, pp. 372–387, IEEE, 2016.

[163] O. Suciu, S. E. Coull, and J. Johns, "Exploring adversarial examples in malware detection," in *2019 IEEE Security and Privacy Workshops (SPW)*, pp. 8–14, IEEE, 2019.

[164] K. Grosse, N. Papernot, P. Manoharan, M. Backes, and P. McDaniel, "Adversarial examples for malware detection," in *European Symposium on Research in Computer Security*, pp. 62–79, Springer, 2017.

[165] gartner. https://www.gartner.com/en/documents/3899783, 2019.

[166] S. Radack, "Federal information processing standard (fips) 199, standards for security categorization of federal information and information systems," tech. rep., National Institute of Standards and Technology, 2004.

[167] C. M. Bishop, *Pattern recognition and machine learning*. springer, 2006.

[168] X. Chen, C. Li, D. Wang, S. Wen, J. Zhang, S. Nepal, Y. Xiang, and K. Ren, "Android hiv: A study of repackaging malware for evading machine-learning detection," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 987–1001, 2019.

[169] A. S. Chivukula and W. Liu, "Adversarial deep learning models with multiple adversaries," *IEEE Transactions on Knowledge and Data Engineering*, vol. 31, no. 6, pp. 1066–1079, 2018.

[170] P. Samangouei, M. Kabkab, and R. Chellappa, "Defense-gan: Protecting classifiers against adversarial attacks using generative models," *arXiv preprint arXiv:1805.06605*, 2018.

[171] N. Carlini, *Evaluation and design of robust neural network defenses*. PhD thesis, UC Berkeley, 2018.

[172] F. Liao, M. Liang, Y. Dong, T. Pang, X. Hu, and J. Zhu, "Defense against adversarial attacks using high-level representation guided denoiser," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1778–1787, 2018.

[173] B. Biggio, B. Nelson, and P. Laskov, "Support vector machines under adversarial label noise," in *Asian conference on machine learning*, pp. 97–112, 2011.

[174] A. D. Joseph, P. Laskov, F. Roli, J. D. Tygar, and B. Nelson, "Machine learning methods for computer security (dagstuhl perspectives workshop 12371)," in *Dagstuhl Manifestos*, vol. 3, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[175] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[176] W. Stallings, *Cryptography and network security, 4/E*. Pearson Education India, 2006.

[177] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction apis," in *25th {USENIX} Security Symposium ({USENIX} Security 16)*, pp. 601–618, 2016.

[178] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1322–1333, 2015.

[179] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 3–18, IEEE, 2017.

[180] B. Biggio, I. Corona, D. Maiorca, B. Nelson, N. Šrndić, P. Laskov, G. Giacinto, and F. Roli, "Evasion attacks against machine learning at test time," in *Joint European conference on machine learning and knowledge discovery in databases*, pp. 387–402, Springer, 2013.

[181] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," *arXiv preprint arXiv:1412.6572*, 2014.

[182] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, Z. B. Celik, and A. Swami, "Practical black-box attacks against machine learning," in *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pp. 506–519, 2017.

[183] P.-Y. Chen, H. Zhang, Y. Sharma, J. Yi, and C.-J. Hsieh, "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models," in *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 15–26, 2017.

[184] W. Xu, Y. Qi, and D. Evans, "Automatically evading classifiers," in *Proceedings of the 2016 network and distributed systems symposium*, vol. 10, 2016.

[185] W. Yang, D. Kong, T. Xie, and C. A. Gunter, "Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 288–302, 2017.

[186] P. Laskov *et al.*, "Practical evasion of a learning-based classifier: A case study," in *2014 IEEE symposium on security and privacy*, pp. 197–211, IEEE, 2014.

[187] B. Biggio, K. Rieck, D. Ariu, C. Wressnegger, I. Corona, G. Giacinto, and F. Roli, "Poisoning behavioral malware clustering," in *Proceedings of the 2014 workshop on artificial intelligent and security workshop*, pp. 27–36, 2014.

[188] I. Rosenberg, A. Shabtai, L. Rokach, and Y. Elovici, "Low resource black-box end-to-end attack against state of the art api call based malware classifiers," *arXiv preprint arXiv:1804.08778*, 2018.

[189] H. Dang, Y. Huang, and E.-C. Chang, "Evading classifiers by morphing in the dark," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 119–133, 2017.

[190] L. Chen, S. Hou, and Y. Ye, "Securedroid: Enhancing security of machine learning-based detection against adversarial android malware attacks," in *Proceedings of the 33rd Annual Computer Security Applications Conference*, pp. 362–372, 2017.

[191] Q. Wang, W. Guo, K. Zhang, A. G. Ororbia, X. Xing, X. Liu, and C. L. Giles, "Adversary resistant deep neural networks with an application to malware detection," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 1145–1153, 2017.

[192] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks,"

in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 582–597, IEEE, 2016.

[193] F. Tramèr, A. Kurakin, N. Papernot, I. Goodfellow, D. Boneh, and P. McDaniel, "Ensemble adversarial training: Attacks and defenses," *arXiv preprint arXiv:1705.07204*, 2017.

[194] B. Nelson, M. Barreno, F. J. Chi, A. D. Joseph, B. I. Rubinstein, U. Saini, C. A. Sutton, J. D. Tygar, and K. Xia, "Exploiting machine learning to subvert your spam filter.," *LEET*, vol. 8, pp. 1–9, 2008.

[195] D. Meng and H. Chen, "Magnet: a two-pronged defense against adversarial examples," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 135–147, 2017.

[196] W. Xu, D. Evans, and Y. Qi, "Feature squeezing: Detecting adversarial examples in deep neural networks," *arXiv preprint arXiv:1704.01155*, 2017.

[197] T. Chakraborty, F. Pierazzi, and V. Subrahmanian, "Ec2: Ensemble clustering and classification for predicting android malware families," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[198] Q. Wang, W. Guo, I. Ororbia, G. Alexander, X. Xing, L. Lin, C. L. Giles, X. Liu, P. Liu, and G. Xiong, "Using non-invertible data transformations to build adversarial-robust neural networks," *arXiv preprint arXiv:1610.01934*, 2016.

[199] L. Chen, Y. Ye, and T. Bourlai, "Adversarial machine learning in malware detection: Arms race between evasion attack and defense," in *2017 European Intelligence and Security Informatics Conference (EISIC)*, pp. 99–106, IEEE, 2017.

[200] A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, and F. Roli, "Yes, machine learning can be more secure! a case study on android malware detection," *IEEE Transactions on Dependable and Secure Computing*, 2017.

[201] F. Zhang, P. P. Chan, B. Biggio, D. S. Yeung, and F. Roli, "Adversarial feature selection against evasion attacks," *IEEE transactions on cybernetics*, vol. 46, no. 3, pp. 766–777, 2015.

[202] D. Li, Q. Li, Y. Ye, and S. Xu, "Enhancing robustness of deep neural networks against adversarial malware samples: Principles, framework, and aics'2019 challenge," *arXiv preprint arXiv:1812.08108*, 2018.

[203] J. W. Stokes, D. Wang, M. Marinescu, M. Marino, and B. Bussone, "Attack and defense of dynamic analysis-based, adversarial neural malware classification models," *arXiv preprint arXiv:1712.05919*, 2017.

[204] L. Tong, B. Li, C. Hajaj, C. Xiao, and Y. Vorobeychik, "A framework for validating models of evasion attacks on machine learning, with application to pdf malware detection," *arXiv preprint arXiv:1708.08327*, 2017.

[205] D. Lowd and C. Meek, "Good word attacks on statistical spam filters.," in *CEAS*, vol. 2005, 2005.

[206] D. Wagner and P. Soto, "Mimicry attacks on host-based intrusion detection systems," in *Proceedings of the 9th ACM Conference on Computer and Communications Security*, pp. 255–264, 2002.

Biographies

ShymalaGowri Selvaganapathy is working as Assistant Professor in the department of Information Technology, PSG College of Technology, India since 2012. Her research interests include Malware Detection, Adversarial Machine Learning, Information Security, Attacks and Defense techniques. She received her M.E. degree in Computer Science and Engineering in the year 2012 and B.Tech degree in Information Technology in the year 2007 from Anna University, India.

G. Sudha Sadasivam is working as Professor and is heading the Department of Computer Science and Engineering in PSG College of Technology, India. She has 24+ years of teaching experience. Her areas of interest include

Distributed Systems, Distributed Object Technology, Grid, Cloud Computing and Security. She has published 80+ research papers in refereed international and national journals, and at conferences. She has published five books in her areas of interest. She has coordinated two AICTE RPS projects in distributed and grid computing arena. She is the coordinator for PSG-Yahoo research on grid and cloud computing.

Vinayakumar Ravi received the Ph.D. degree in computer science from Computational Engineering & Networking, Amrita School of Engineering, Coimbatore, Amrita Vishwa Vidyapeetham, India. He is currently Assistant Research Professor at Center for Artificial Intelligence, Prince Mohammad Bin Fahd University, Khobar, Saudi Arabia. Prior to that, he was a Postdoctoral research fellow in developing and implementing novel computational and machine learning algorithms and applications for big data integration and data mining with Cincinnati Children's Hospital Medical Center, Cincinnati, OH, USA. He has worked on various Cyber Security problems such as intrusion detection, malware detection, ransomware detection, DGA analysis, network traffic analysis, botnet detection, spam and phishing detection in email and URL, image spam detection, and spoofing detection. He has more than 50 research publications in reputed IEEE conferences, IEEE Transactions and Journals.